贪心算法

内容

- 贪心算法的基本概念
- 贪心算法获得最优解的基本条件
  - 最优子结构
  - 贪心选择性
- 应用贪心算法解决
  - 活动安排问题
  - 最优装载问题
  - 哈夫曼编码问题
  - 单源最短路径问题
  - 最小生成树问题
  - 多机调度问题

贪心算法的基本概念

- 贪心算法
  - 解决优化问题
  - 策略：逐步解决问题。总是作出当前看起来最好的选择，即局部最优解

贪心算法的基本概念

- 例1
  - 给定4种硬币（五毛、一毛、五分、一分）
  - 用最少的硬币找顾客n毛n分（六毛三分）
- 贪心算法
  - 当前看起来最优的选择（局部最优解）：每次选不超过余额的面值最大的硬币

得到的结果是一个整体最优解

贪心算法的基本概念

- 例2
  - 给定3种硬币（一毛一、五分、一分）
  - 用最少的硬币找顾客n毛n分（一毛五）
- 贪心算法
  - 1个一毛一
  - 4个一分
- 最优解
  - 3个五分

不是整体最优解

贪心算法的基本概念

- 贪心算法何时可以得到（整体）最优解？
  - 优化问题需具有
    - 贪心选择性
    - 最优子结构

活动安排问题

活动安排问题

- 输入
  - n个活动的集合$E=\{1, 2, \ldots, n\}$
  - 每个活动 i 的持续时间 $[s_i, f_i)$
    - 所有活动使用同一资源
    - 同一时刻不能有两个活动使用该资源
    - 如果 $s_i \geq f_j$ 或 $s_j \geq f_i$，则活动 i 与活动 j 相容
- 输出
  - 活动集合中最大的相容活动子集

活动安排问题

- 贪心算法
  - 假设各个活动按活动结束时间 f_i 排序
    - $f_1 \leq f_2 \leq \ldots \leq f_n$
  - 选择活动 **1** (结束时间最早的活动)
  - 从2开始按顺序考察各个活动，选择第一个与活动 **1** 相容的活动 $\boldsymbol{i}$
  - 从 i+1开始按顺序考察各个活动，选择第一个与活动 $\boldsymbol{i}$ 相容的活动 $\boldsymbol{j}$
  - 。。。

每次选择与现有活动相容的结束时间最早的活动

活动安排问题

- 贪心算法

```
void GreedySelector(int n, Type s[], Type f[], bool A[])
{
      A[1]=true;
      int j=1;
      for (int i=2;i<=n;i++) {
        if (s[i]>=f[j])  { A[i]=true;    j=i; }
        else        A[i]=false;
        }
}
```

时间复杂性$O(n)$

活动安排问题

i	1	2	3	4	5	6	7	8	9	10
s[i]	1	3	0	5	3	5	6	8	8	11
f[i]	4	5	6	7	8	9	10	11	12	14
A[i]	✓	✗	✗	✓	✗	✗	✗	✓	✗	✓

活动安排问题

- 证明贪心算法可以得到最优解（归纳法）
  - 证明第一次选择活动1是正确的
    - 即活动1在最优解中
  - 证明选择完活动1后，问题变成了输入为

    E'={与活动1相容的活动}

    的子问题
    - 因为第二个选择的活动 i 是 E' 中结束时间最早的，所以活动 i 是正确的
    - 依次类推所有的选择都是正确的

活动安排问题

- 证明活动1在最优解中
  - 假设 $A \subseteq E$ 是活动安排问题的一个最优解，
    - A中结束时间最早的活动为k
  - 如果k=1，活动1在最优解中
  - 如果k>1，$B = (A - \{k\}) \cup \{1\}$ 也是一个最优解

贪心选择性

活动安排问题

- 证明选择完活动1后，问题变成了输入为

  E'={与活动1相容的活动}

  的子问题
  - 假设 $A \subseteq E$ 是活动安排问题的一个最优解，且包含活动1
  - 于是A'=A-{1}是针对**E'**的活动安排问题的最优解
    - 否则，假设E'中有更优解B
    - B+{1}是针对E的一个更优解

最优子结构

贪心算法获得最优解的基本条件

- 贪心选择性
  - （第一次）作出贪心选择是正确的
- 优化子结构
  - （第一次）做完贪心选择后，得到一个与原问题定义相同（输入不同）的子问题

最优装载问题

- 输入
  - n个集装箱：集装箱i的重量为w_i
  - 轮船的载重量C
- 输出
  - （尽可能多）装入轮船的集装箱
- 形式化

$$\max \sum_{i=1}^{n} x_i$$

s.t.

$$\sum_{i=1}^{n} w_i x_i \le C$$

$$x_i \in \{0, 1\}, \quad 1 \le i \le n$$

最优装载问题

- 贪心算法
  - 逐个选择集装箱装入轮船
  - 每次选择最轻的集装箱

```
void Loading(int x[],  Type w[], Type C, int n)
    {
        int *t = new int [n+1];
        Sort(w, t, n);
        for (int i = 1; i <= n; i++) x[i] = 0;
        for (int i = 1; i <= n && w[t[i]] <= C; i++)
          {  x[t[i]] = 1;    C -= w[t[i]];}
    }
```

时间复杂性：$O(n \log n)$

最优装载问题

- 设集装箱已经依其重量从小到大排 $w_1 \le w_2 \le .... \le w_n$
- 贪心选择性
  - 第一步选择第一个（最轻的）集装箱是正确的，即第一个集装箱一定在最优解中
  - 设最优解选择的集装箱为{a, b, c, …}（按重量从小到大排列）
  - 如果a=1, 则最轻的集装箱在最优解中
  - 如果a>1，{1, b, c, …}同样为问题的最优解

最优装载问题

- 设集装箱已依其重量从小到大排 $w_1 \le w_2 \le .... \le w_n$
- 优化子结构
  - 设问题的最优解为{1, b, c, …, }（按重量从小到大排列）
  - 则{b, c, …, }针对以下输入的最优装载问题的最优解
    - 集装箱{2, …, n}
    - 轮船载重C-w_1

最优装载问题

- 结论
  - 贪心算法可以获得最优装载问题的最优解

哈夫曼编码

❑ 对字符编码

- 10,000个字符：对每个字符用0,1编码

	a	b	c	d	e	f
Frequency	4500	1300	1200	1600	900	500
Codes	**000**	**001**	**010**	**011**	**100**	**101**

- 定长编码
  - 编码长度：10,000 * 3=30,000

哈夫曼编码

❑ 对字符编码

- 10,000个字符：对每个字符用0,1编码

	a	b	c	d	e	f
Frequency	4500	1300	1200	1600	900	500
Codes	**0**	**101**	**100**	**111**	**1101**	**1100**

- 变长编码
  - 编码长度：4500*1+1300*3+1200*3+1600*3+900*4+500*4=22400

哈夫曼编码

- 前缀码
  - 每个字符规定一个0,1串作为编码
  - 任意字符的编码都不是其它字符编码的前缀
  - 译码简单，只需要按顺序取出代表某一字符的前缀码

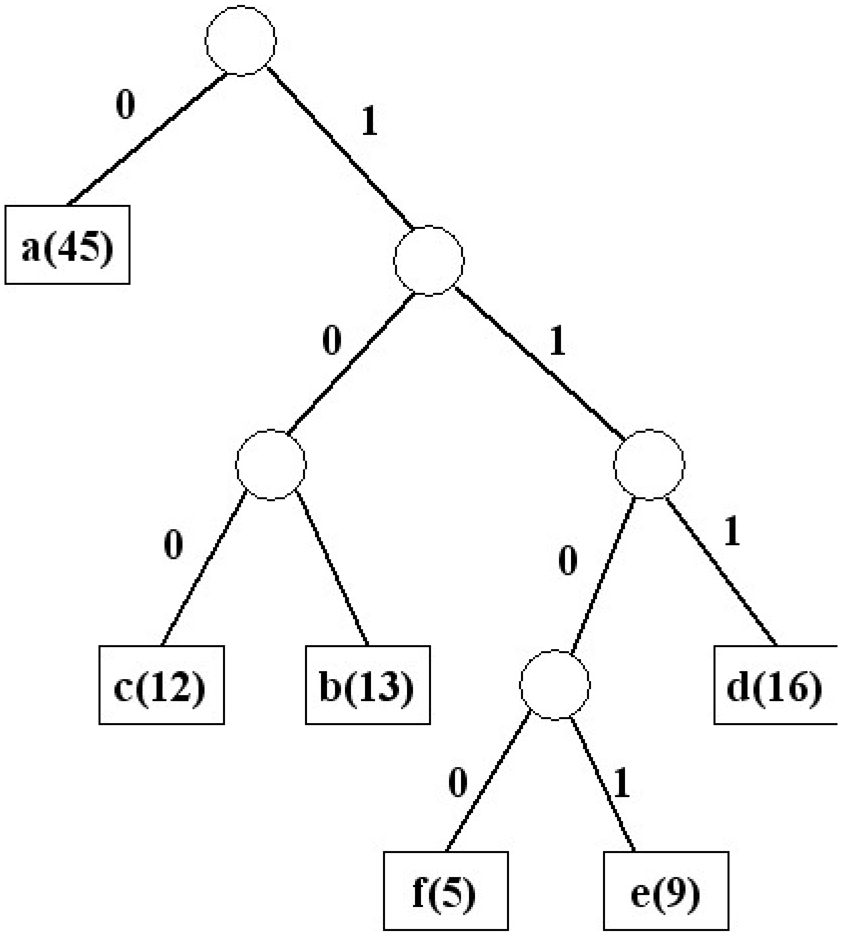
	a	b	c	d	e	f
Frequency	4500	1300	1200	1600	900	500
Codes	**0**	**101**	**100**	**111**	**1101**	**1100**

- 接收：001011101
- 解码：aabe

为了更方便的取出编码前缀，需要一个合适的数据结构来表示前缀码，为此，可以用二叉树来表示。

	a	b	c	d	e	f
Frequency	4500	1300	1200	1600	900	500
Codes	**0**	**101**	**100**	**111**	**1101**	**1100**

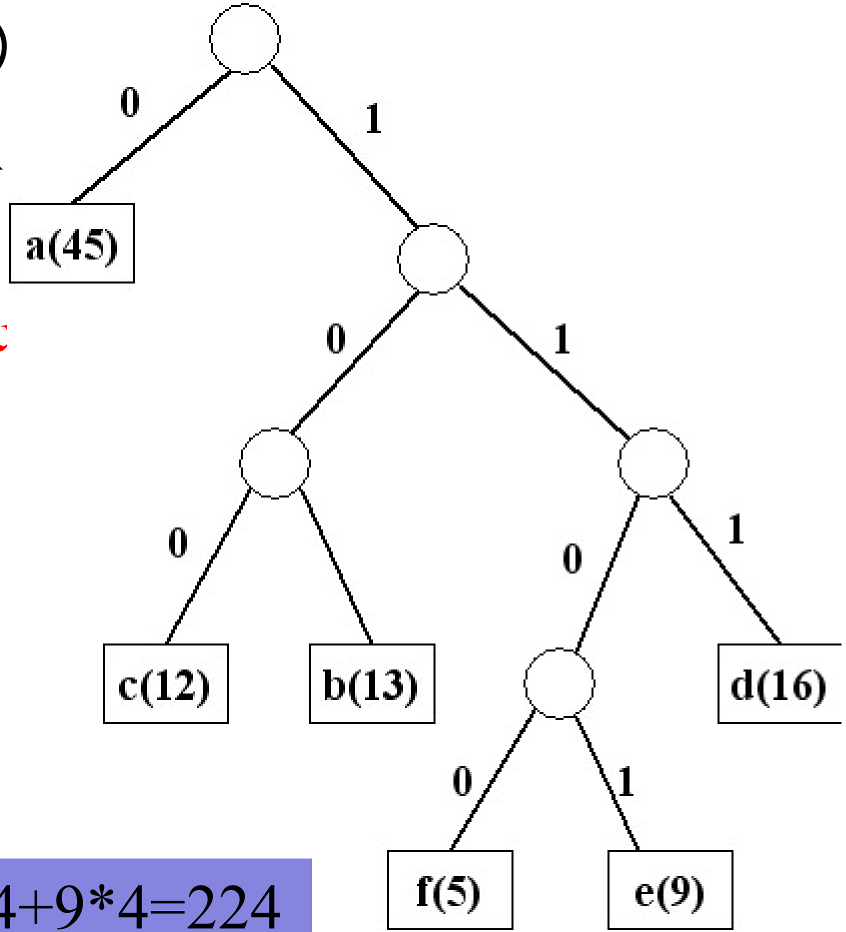
- 前缀码←→二叉树
  - 左分支：0
  - 右分支：1
  - 树叶代表字符
  - 从树根到树叶的路径代表字符编码



哈夫曼编码

- 平均码长（二叉树代价）
  - 一颗根据字符集C构造的二
  - 对于C中的任意字符x
    - 其出现频率（权重）为$f($
    - 其在T中的深度为$d_T(x)$
  - 则T的平均码长（代价为）

$$B(T)=\sum_{x\in C} f(x)d_T(x)$$



$B(T)$=45*1+12*3+13*3+16*3+5*4+9*4=224

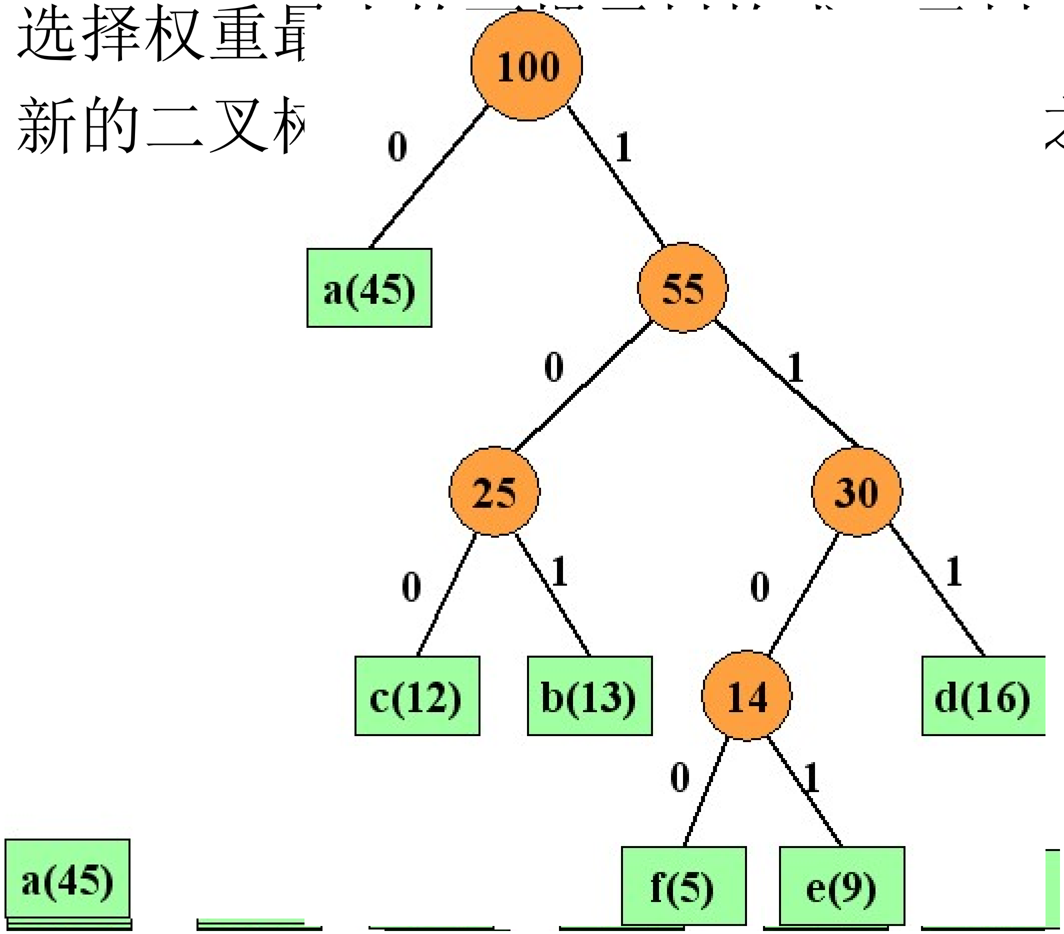
哈夫曼编码问题

- 输入
  - 字符集C，对于C中的任意字符x，其出现频率（权重）为$f(x)$
- 输出
  - 平均码长最短的前缀码编码方案（哈夫曼编码）
    - 即代价最小的前缀二叉树

哈夫曼编码问题

贪心算法

- 选择权重最
- 新的二叉 之和



哈夫曼编码问题

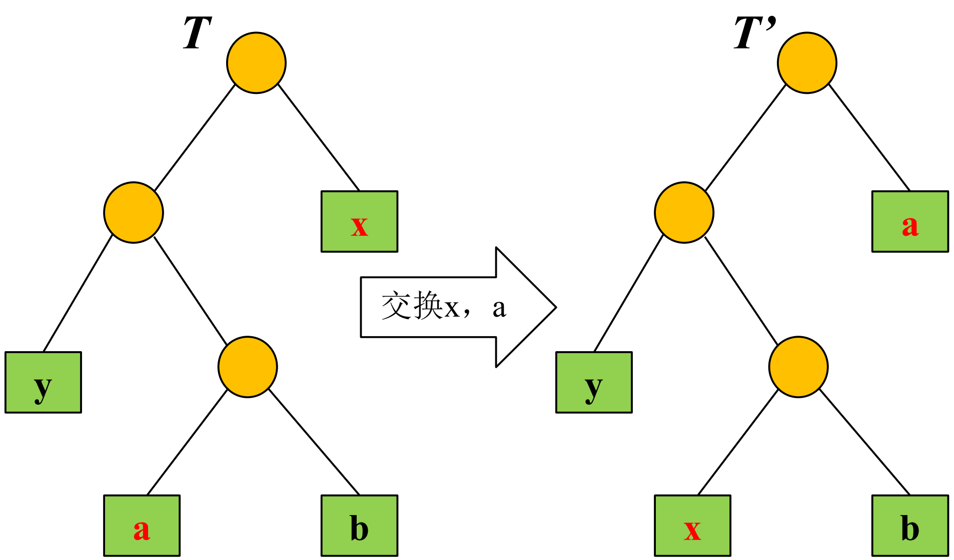
- 贪心算法
  - 建立一个由所有字符构成的堆Q
  - 循环执行
    - 取（删除）Q中的堆顶元素x
    - 取（删除）Q中的堆顶元素y
    - 将x和y合并为二叉树z，其权值为x和y的权值之和
    - 将z插入Q中

每次堆操作需要$O(\log n)$时间
共n-1次合并操作
时间复杂性：$O(n \log n)$

哈夫曼编码问题

贪心选择性

- 设x和y是给定字符集中权重最小的两个字符，在最优二叉树T中，x和y一定是最深的叶子且互为兄弟
- 证明：如果不是这样

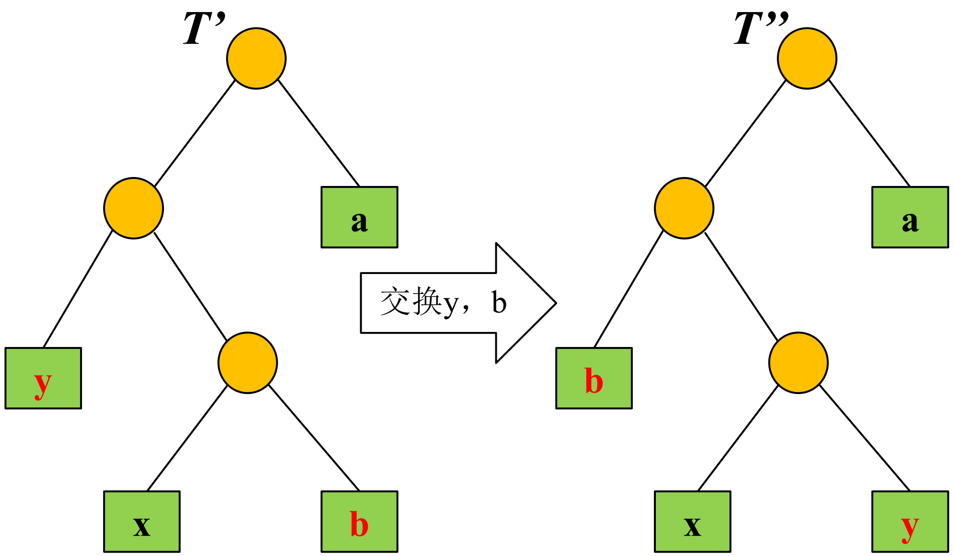


$$
\begin{aligned}
&B(T)-B(T')\\
&=\sum_{c\in C} f(c)d_T(c)-\sum_{c\in C} f(c)d_{T'}(c)\\
&=f(x)d_T(x)+f(a)d_T(a)-f(x)d_{T'}(x)-f(a)d_{T'}(a)\\
&=f(x)d_T(x)+f(a)d_T(a)-f(x)d_T(a)-f(a)d_T(x)\\
&=(f(a)\text{-}f(x))(d_T(a)-d_T(x))\geq 0
\end{aligned}
$$

代价不增加！

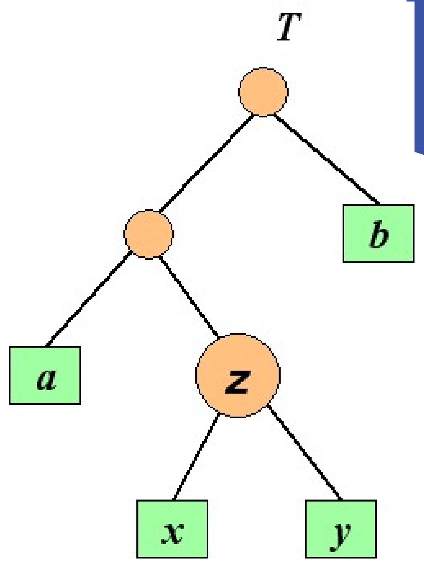
哈夫曼编码问题

- 贪心选择性
  - 设x和y是给定字符集中权重最小的两个字符，在最优二叉树T中，x和y一定是最深的叶子且互为兄弟
  - 证明：如果不是这样

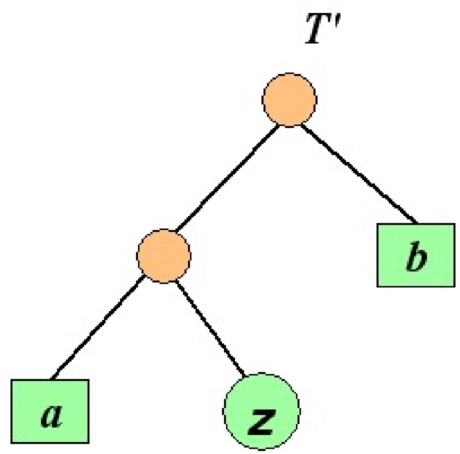


同理，T'变换到T''，同样不增加代价。如果T是最优的，那么T''一定是最优的，x，y是最深的叶子。

哈夫曼编码问题



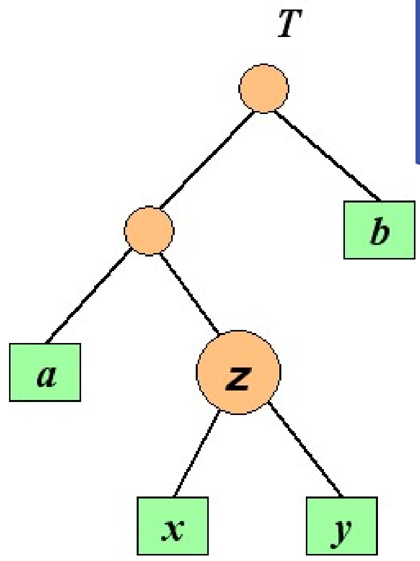
- 最优子结构
  - 设x和y是给定字符集C中权重最小的两个字符
  - 在最优二叉树T中，x和y是两个最深的叶子且互为兄弟
  - 设z是x和y的父亲，将z看作一个新的字符，权重为f(z) = f(x) + f(y)
  - 要证明：**T’=T - {x, y}**是针对字符集**C’= C–{x, y}+{z}**的最优前缀二叉树(证明使T最优的话，**T’**应该最优)
  - 原问题：a，x，y，b
  - 做完选择后，子问题：a，z，b

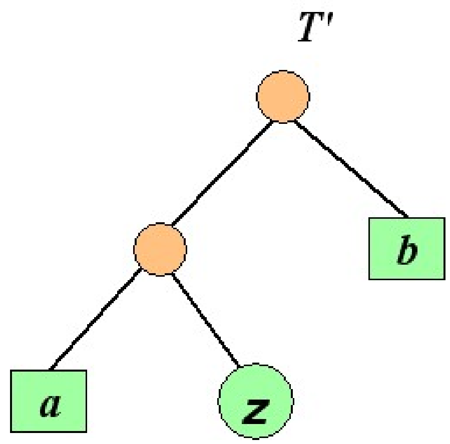


哈夫曼编码问题

- 对于C-{x, y}中的字符a
  - $f(a)\ d_T(a) = f(a)\ d_{T'}(a)$
- 计算B(T')时对于z
  - $f(z)\ d_{T'}(z)$
- 计算B(T)时对于x和y
  - $f(x)\ d_T(x) + f(y)\ d_T(y)$
- **B(T)=B(T')+f(x)+f(y)**

T'=T - {x, y}是针对字符集**C'= C–{x, y}+{z}**的最优前缀二叉树





哈夫曼编码问题

- 结论
  - 贪心算法可以获得哈夫曼编码问题的最优解

贪心算法的证明

- ❐ 贪心选择性

✓ 证明求解过程中的选择都是正确的。

- 活动安排：每次都选择结束时间最早的相容活动，证明当前选择的结束最早的那个相容活动必然在最优解中。
- 装载问题：每次都选择没有装上船的最轻的那个物品，证明当前选择的那个最轻的物品必然在最优解中。
- 哈夫曼编码：每次都选择权重（频率）最小的两个节点作为二叉树的两个分支，并使得其父节点为其权重和，使用堆操作进行删除和插入。需要证明在最优二叉树中，权重最小的两个节点必然为最深的叶子并互为兄弟。

贪心算法的证明

- 最优子结构

✓ 证明最优解包含子问题的最优解

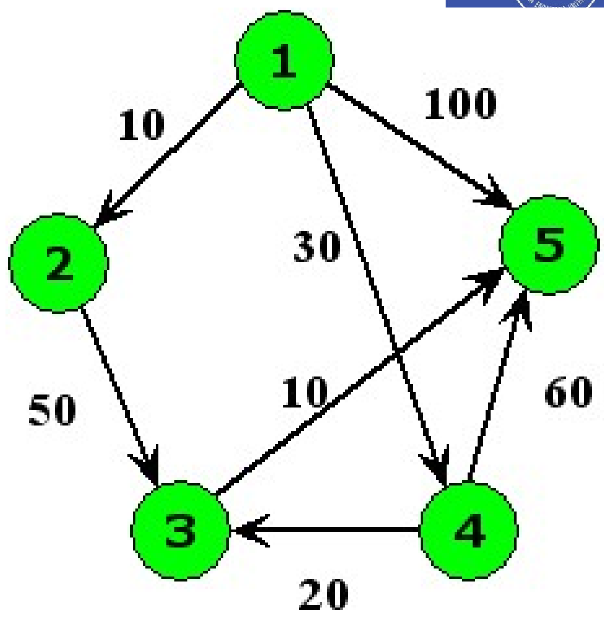
- 一般利用反证法，先假设给出当前问题的最优解，其中包含确定在最优解中的部分和子问题，假设子问题有更好的解，推导出原问题有更优的解。即证明出，原问题的最优解等于确定在最优解中的部分加上子问题的最优解。

贪心算法与动态规划

- 贪心算法：贪心选择性+最优子结构
- 动态规划：最优子结构+重叠子问题
- 动态规划每次求解依赖子问题的求解。
- 贪心算法在当前状态下作出最好的选择得到局部最优解，然后再去解选择之后产生的子问题。
- 动态规划自底向上求解，贪心算法自顶向下求解。

单源最短路径

- ❑ 输入
  - ❍ 有向带权图$G=(V, E)$
    - 对于E中的任意一条边e，其长度为$c(e)$
  - ❍ V中的一个顶点t——源
- ❑ 输出
  - ❍ 图中t到每个顶点的最短路径长度(边权和)



单源最短路径

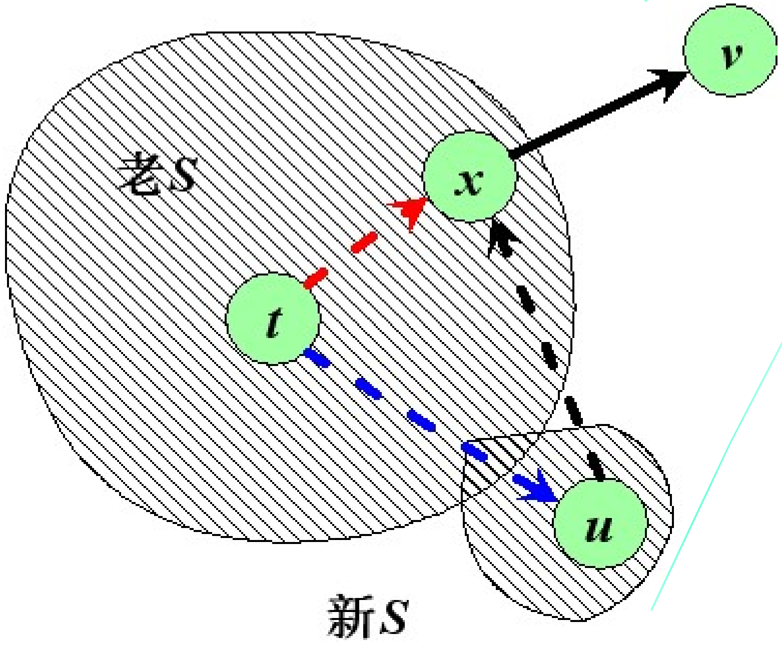
- 贪心算法（Dijkstra算法）
  - 设置集合S来保存所有（t到其）最短路径长度已知的顶点，初始时S={t}
  - 用***dist*(*v*)**来记录t到v的最短特殊路径的长度
    - 如果从t到v的路径中间只经过S中的顶点，这样的路径叫做特殊路径，初始时
    - ***dist*(*v*)** =c(t, v) 如果存在边(t, v)，其中c表示边权
    - ***dist*(*v*)** =INFINITY 如果不存在边(t, v)
  - 算法每次从$V - S$中找出***dist***最小的顶点u，
    - 将u加入S中
    - 更新$V - S$中其它顶点v的***dist*(*v*)**
      - 如果 ***dist*(*u*)** + $c(u, v)$ < ***dist*(*v*)**，则更新***dist*(*v*)**

单源最短路径

- 贪心算法（Dijkstra算法）

为何不用更新S里的顶点？

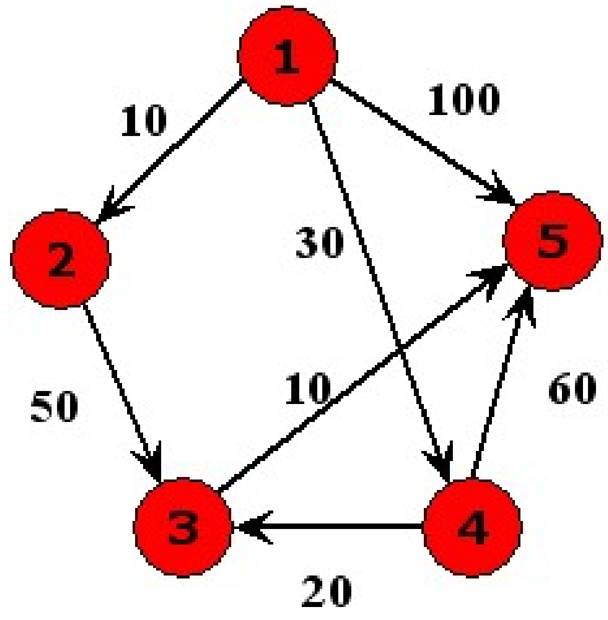
假设当前t到u最小，那么将u加入到S中后，不会使得t到x的路径更短。



由于x先于u加入到S中，那么u加入到S之前，t到x一定小于t到u，当u加入到S中后，t经过u再到x的路径一定大于t到x的当前最短路径。

单源最短路径

- Dijkstra算法
  - t=1



迭代	S	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	10	+∞	30	100
1	{1, 2}	10	60	30	100
2	{1, 2, 4}	10	50	30	90
3	{1, 2, 4, 3}	10	50	30	60
4	{1, 2, 4, 3, 5}	10	50	30	60

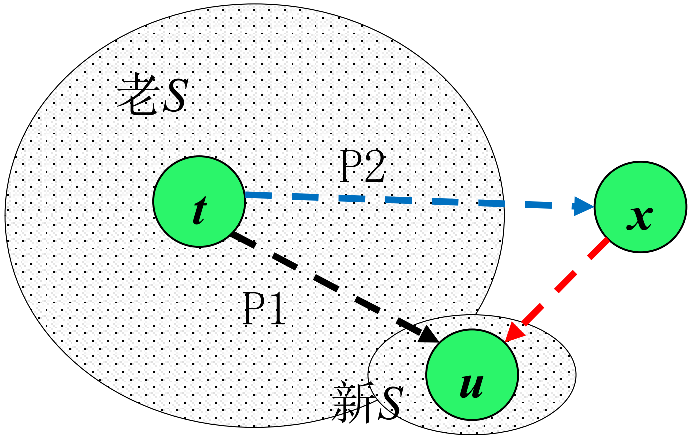
单源最短路径

- 时间复杂性
  - S被扩充n-1次
  - 每次扩充选择u需要$O(n)$时间
  - 每次扩充更新节点的***dist***(v)需要$O(n)$时间
  - 总时间：$O(n^2)$

单源最短路径

贪心选择性

- 从$V - S$中选择***dist***最小的顶点u加入到S中是正确的.
  - 即从t到u的最短特殊路径就是从t到u的最短路径
  - 即t到V-S其它顶点再到u的路径比最短的特殊路径短



因为对于每次选择u加入S，t到u小于t到任意V-S中的x，即P1<P2。即：P1为t到u的最短特殊路径，同时小于任何其它经过V-S里顶点再到达u的路径，即最短路径，加入正确。

查看经过V-S的点

单源最短路径

❑ 最优子结构

问题：一条最短路径

- 如果P(i,j)={Vi....Vk..Vs...Vj}是从顶点i到j的最短路径，k和s是这条路径上的一个中间顶点，那么P(k,s)必定是从k到s的最短路径。
- 证明：

假设P(i,j)={Vi....Vk..Vs...Vj}是从顶点i到j的最短路径，则有P(i,j)=P(i,k)+P(k,s)+P(s,j)。而P(k,s)不是从k到s的最短距离，那么必定存在另一条从k到s的最短路径P'(k,s)，那么P'(i,j)=P(i,k)+P'(k,s)+P(s,j)<P(i,j)。则与P(i,j)是从i到j的最短路径相矛盾。

单源最短路径

最优子结构

问题：单源最短路径（选择vi加入S）

S中顶点最短路径已知，V-S中顶点已知当前最短特殊路径长度

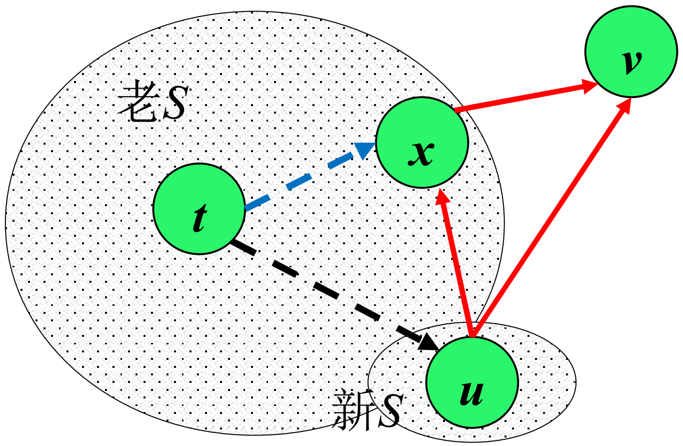
- 原问题：S={t,v1,v2,...,vi-1},V-S={vi,vi+1,...vn}
- 新问题：S={t,v1,v2,...,vi},V-S={vi+1,...vn}
- Dist(vi) 就是最短路径已经证明.
- 很明显：新问题的解dist(vi+1...vn) 如果最优，那么它一定是原问题的最优解。
- 需要满足：加入vi后，V-S里的dist更新正确，即加入后确实比加入前更优。

单源最短路径

如果 $dist(u) + c(u, v) < dist(v)$，则更新$dist(v)$

最优子结构

- 设u加入S之前（老S）$V - S$中每个顶点v的$dist(v)$确实是 t 到v的最短特殊路径长度
- 要证明：每次向S新加入u之后（新S），更新$V – S$中其它顶点v的$dist(v)$是正确的
  - 即更新后的$dist(v)$确实是t到v的最短特殊路径长度



查看经过S的点的路径

如果对应到一条路径问题：
t到v的最短特殊路径一定包含u到v的最短特殊路径。

u加入到S，对于v来说最多增加两类特殊路径：
p1： tu+边c（u，v）
p2： tu+uxv （如果存在）
原dist(v)： txv（如果存在）
由于：x先于u加入，tx<tu+ux，所以原dist(v) <p2
如果： p1<原 $dist(v)$，那么p1<p2, **uv<uxv**
即此时更新值**dist(v)**是最优的

单源最短路径

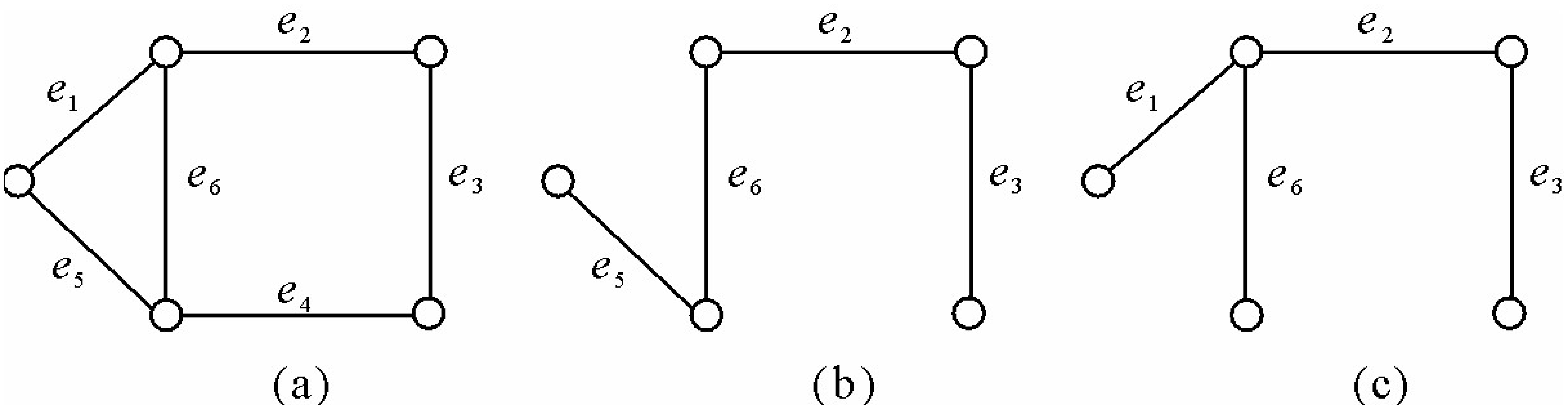
结论

- Dijkstra算法可以获得单源最短路径问题的最优解

最小生成树

生成树

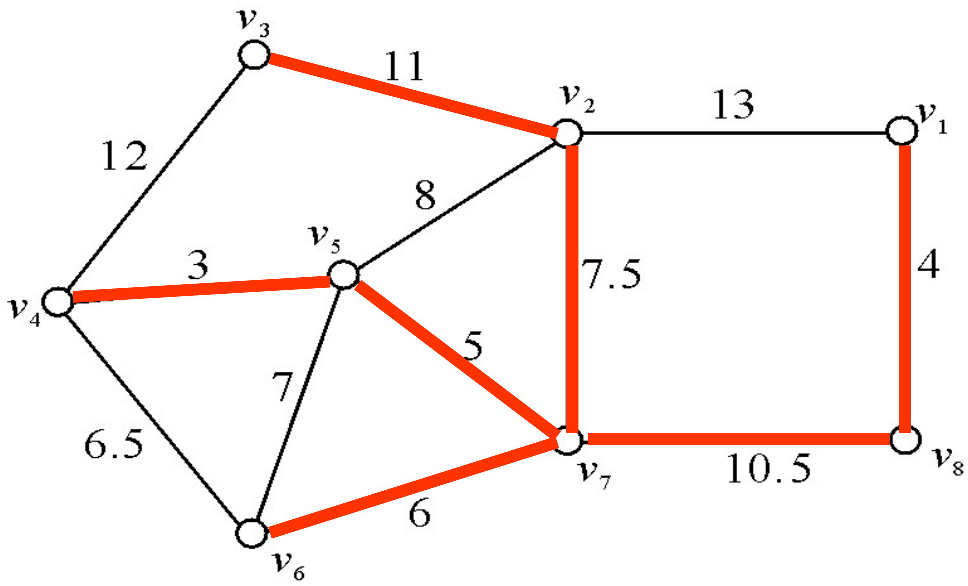
- 对于无向图G=(V, E)，如果G的子图T包含了G中的所有顶点且T是一棵树，则T称为G的生成树



最小生成树

❑ 最小生成树问题

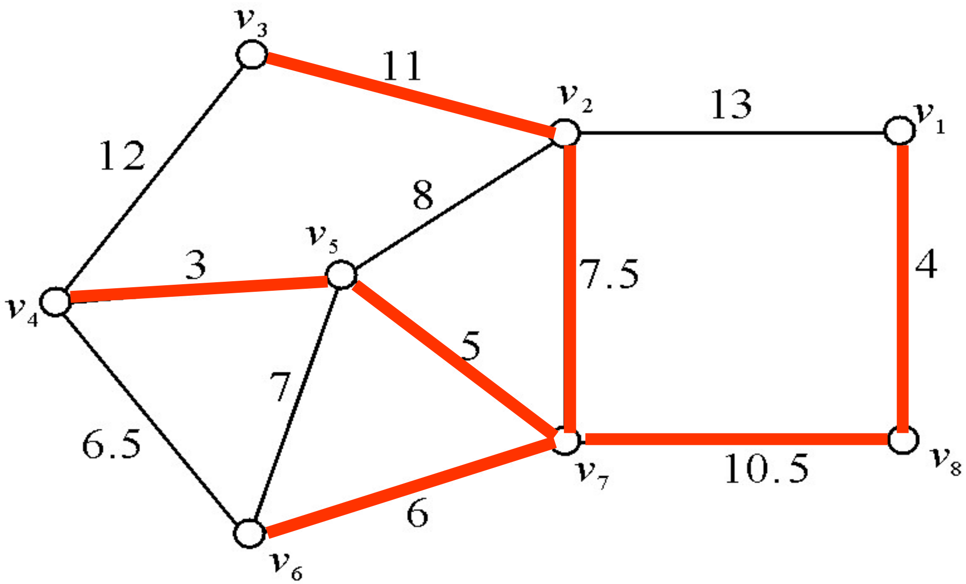
- 输入：无向带权图G=(V, E)
  - 对于G中的任意一条边e，其权值为$c(e)$
- 输出：G的**最小生成树**T
  - 在所有生成树中T的权值最小
  - T的权值为T中所有边的权值之和



最小生成树

❑ 应用

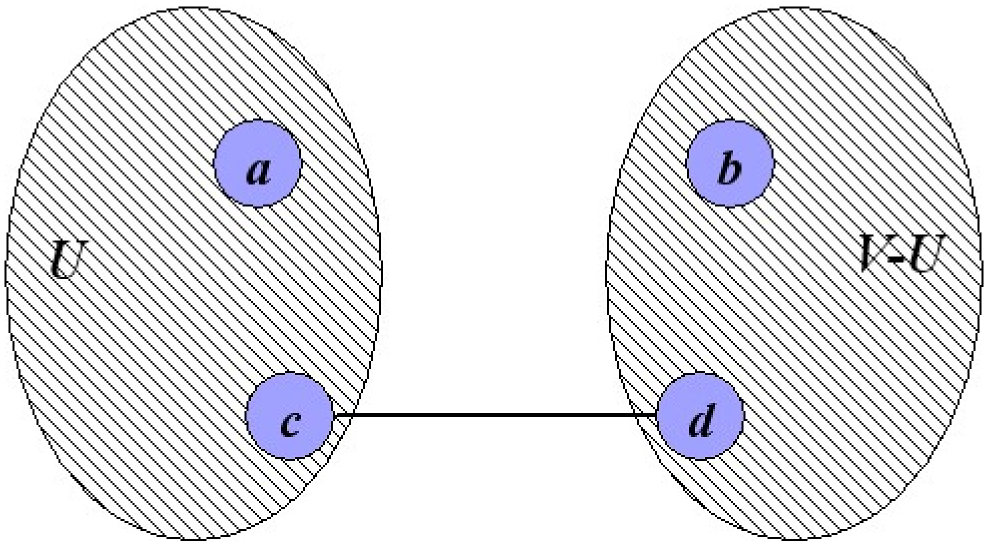
- 用图的顶点代表城市，用边权代表城市间通信线路的成本，最小生成树给出最经济的方案



最小生成树

❑ 最小生成树的性质

- 给定图G=(V, E)，设U是V的真子集
- 设边(a, b) 是所有连接U和V-U的边中权值最小的边
  - $a \in U, b \in V-U$
- **结论：G的最小生成树中一定包含边(a, b)**



最小生成树

- **Prim算法：**
  - 输入：无向连通带权图G=<V,E>
  - 输出：G的最小生成树

1. 取G中的任意节点v_0，T={v_0}。

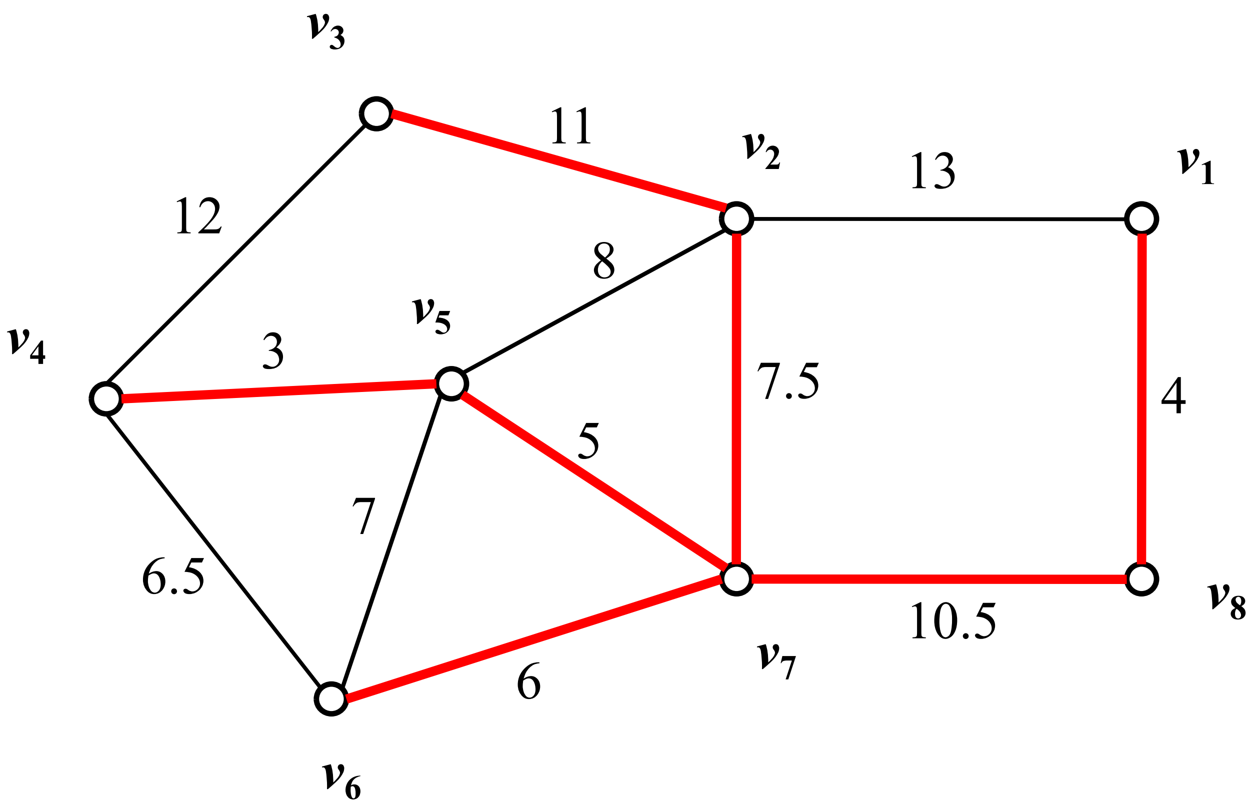
2. 找到权值最小的边(a, b)满足$a \in$T, $b \in$V-T。

3. T=T∪{b}

4. 反复做第2、3步直到所有节点都加入T中。

最小生成树

- Prim算法



最小生成树

- Prim算法
  - 总是寻找连接T和V-T的权值最小的边加入树中
  - Prim算法输出的是最小生成树

最小生成树

1. 取G中的任意节点v_0，T={v_0}。

2. 找到权值最小的边(a, b)满足$a \in T$, $b \in V-T$。

3. T=T∪{b}

4. 反复做第2、3步直到所有节点都加入T中。

Prim算法的复杂性

- 对于给定的图G=(V, E)，n=|V|, m=|E|
- 第2步可以在O(n)的时间内完成
- 复杂性：$O(n^2)$

最小生成树

$G=(V, E)$， $n=|V|$, $m=|E|$

❑ Kruskal算法

1. 初始时T包含图中的n个顶点（没有边）
2. 将图中的边按权值大小排序
3. 逐条考察每条边(u, v)
   - 如果(u, v)连接T中的两个不同的分支，则向T中添加(u, v)

最小生成树

Kruskal算法

① 先将m条边按权由小到大排列：

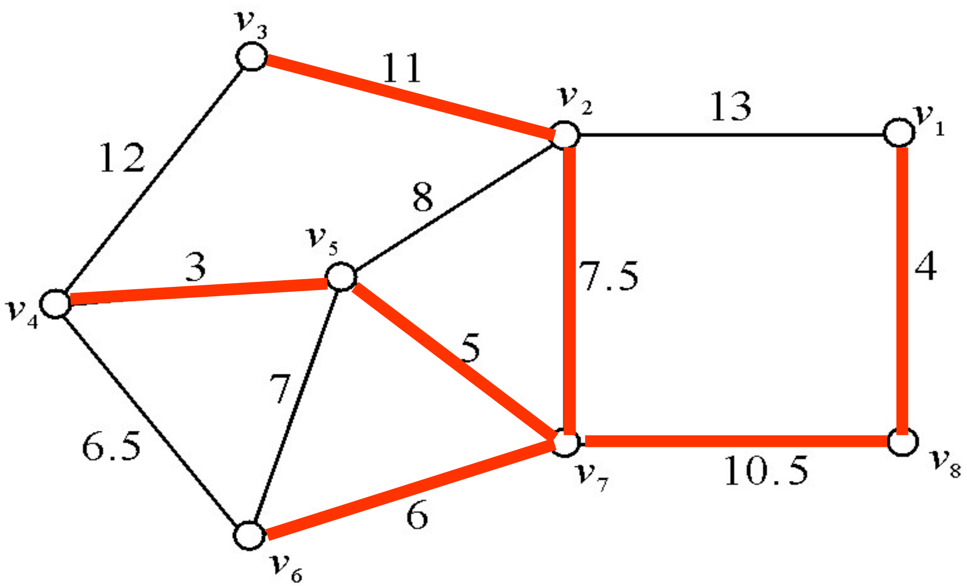
(v_4,v_5)，(v_1,v_8)，(v_5,v_7)，
(v_6,v_7)，(v_4,v_6)，(v_5,v_6)，
(v_2,v_7)，(v_2,v_5)，(v_7,v_8)，
(v_2,v_3)，(v_3,v_4)，(v_1,v_2)。

它们的权分别是：

3，4，5，6，6.5，7，
7.5，8，10.5，11，12，13

② 逐次取边：

(v_4,v_5)，(v_1,v_8)，(v_5,v_7)，
(v_6,v_7)，(v_2,v_7)，(v_7,v_8)，
(v_2,v_3)



最小生成树

- Kruskal算法
  - 总是选择连接T的某个分支和其它节点的权值最小的边加入树中
  - Kruakal算法输出的是最小生成树

最小生成树

Kruskal时间复杂性

- 将边排序需要$O(m\log m)$时间
- 第3步每次循环中判断(u, v)是否属于两个不同分支所需时间为$O(\log n)$
- 第3步总时间为$O(m\log n)$
- 时间复杂性：$\boldsymbol{O(m\log m)}$

G=(V, E)，n=|V|, m=|E|

1. 初始时T包含图中的n个顶点
2. 将所有边按权值大小排序
3. 逐条考察每条边(u, v):

   如果(u, v)连接T中的两个不同的分支，则向T中添加(u, v)

多机调度问题

- 输入
  - n个独立的作业$\{1, 2, \ldots, n\}$
    - 作业 i 所需的处理时间为 t_i
  - m台机器
    - 任何作业可以在任何机器上完成
    - 作业处理不允许中断
- 输出
  - 最优作业调度方案
    - 所有作业在最短时间内完成

NP难问题:还没有多项式时间算法

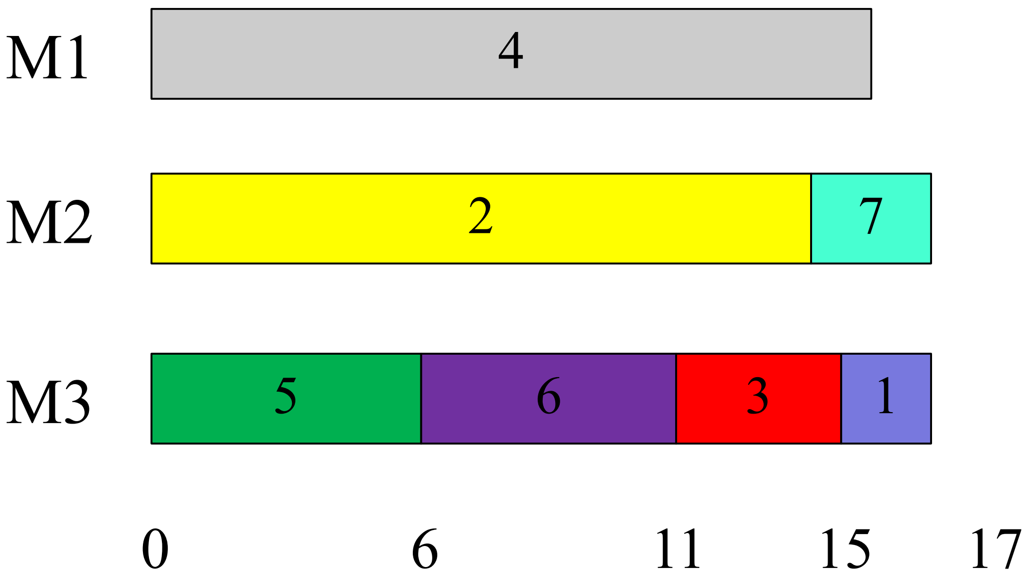
多机调度问题

- n<m时
  - 每个作业分配一台机器
- n>m时：贪心算法
  - 将所有作业按处理时间从大到小排列
  - 按顺序将每个作业分配给最先空闲的机器

多机调度问题

- 输入（三台机器）

Job	4	2	5	6	3	7	1
Time	16	14	6	5	4	3	2



多机调度问题

- 贪心算法时间复杂性
  - 排序$O(n \log n)$
  - 每个作业选择最早空闲的机器耗时$O(\log m)$
  - 总耗时$O(n\log n + n\log m) = O(n\log n)$

多机调度问题

- 近似比
  - 算法的解代价为C
  - 最优解代价为C*
  - 如果$C / C^* \leq a$，则算法是近似比为a的算法

多机调度问题

❑ 贪心算法的近似比

- 作业已经按处理时间排好序
- 最优解的代价（完成时间）

$$T^* \geq \max\{\frac{\sum_{i=1}^{n} t_i}{m}, t_1\}$$

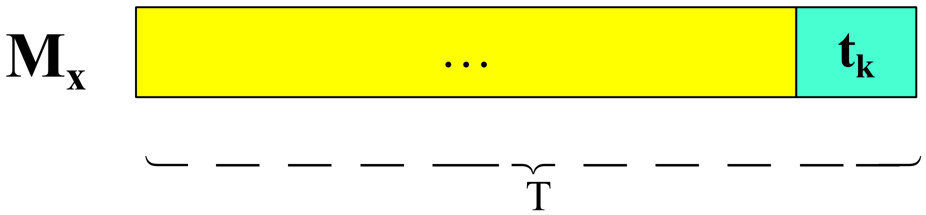
- 即：假设一个任务可以同时分在两个以上的机器上，那么将n个任务的完成时间总和除以m，是这n个作业在这m个机器上完成时间的最小值。但实际上，任务不能分割，最优解肯定大于等于t1。所以最优解是大于等于均值和t1的较大的那个。

多机调度问题

最优解的代价（完成时间）

$$T^* \geq \max\{\frac{\sum_{i=1}^{n} t_i}{m}, t_1\}$$

- 贪心算法的解的代价为T
  - 机器 M_i 的总处理时间为 T_i
  - T 为 M_x 的处理时间
- 如果$t_k=t_1$，$T=T^*=t_1$
- 如果$t_k \neq t_1$：
  - 对于$M_i \neq M_x$，有$T_i \geq T- t_k$
  - 且T- $t_k \geq t_k$ （排序）
  - 所以，$T_i \geq T/2$ （$t_k <= T/2$)



$$\sum_{i=1}^{n} t_i = \sum_{i=1}^{m} T_i \geq \frac{m}{2} T$$

$$T^* \geq \frac{\sum_{i=1}^{n} t_i}{m} \geq \frac{T}{2}$$

多机调度问题

- 结论
  - 贪心算法的近似比为2

第四章完